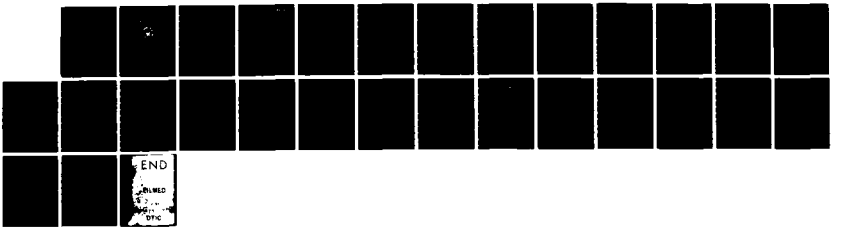
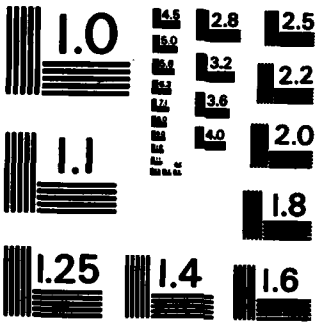
AD-A148 184 TOWARDS AN IDEAL DATABASE SERVER FOR OFFICE AUTOMATION ENVIRONMENTS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S A DEMURJIAN ET AL. OCT 84 NPS52-84-018 1/1

UNCLASSIFIED F/G 9/2 NL

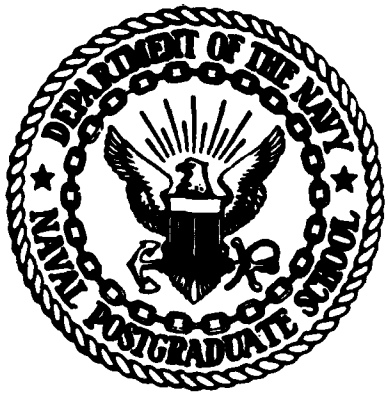




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NPS52-84-018

NAVAL POSTGRADUATE SCHOOL
Monterey, California



TOWARDS AN IDEAL DATABASE SERVER FOR OFFICE AUTOMATION ENVIRONMENTS

Steven A. Demurjian, David K. Hsiao, Douglas S. Kerr, and Paula R. Strawser

October 1984

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by Contract N00014-84-WR-24058 from the Office of Naval Research

This report was prepared by:

DAVID K. HSIAO
Professor of Computer Science

Reviewed by:

BRUCE J. MACLENNAN, Acting Chairman
Department of Computer Science

KNEALE T. MARSHALL
Dean of Information and Policy Sciences

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-84-018	2. GOVT ACCESSION NO. AD-A148 184	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Towards An Ideal Database Server For Office Automation Environments		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Steven A. Demurjian, David K. Hsiao, Douglas S. Kerr and Paula R. Strawser		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N; RR014-08-01 N0001484WR2405°
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, Virginia 22217		12. REPORT DATE October 1984 13. NUMBER OF PAGES 27
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Office automation systems are growing, both in use and in complexity. The development of a database management system for the office automation environment becomes a high priority, in order to provide an efficient and reliable way to manage the information needs of the office. Therefore, the specification of an 'ideal' database server for the office automation environment becomes a key area of concern. In addition to providing traditional support, the ideal database server must also provide new database support, in order to meet the unique and many needs of office automation environments. In this paper, we focus on the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

characterization and specification of an ideal database server for the office automation environment. We also consider how such an ideal database server can be effectively integrated into the office automation environment. Further, we examine an experimental database system, known as the multi-backend database system (MBDS), as a candidate for the ideal database server in the office automation environment.

TOWARDS AN IDEAL DATABASE SERVER FOR OFFICE AUTOMATION ENVIRONMENTS *

Steven A. Demurjian, David K. Hsiao,
Douglas S. Kerr and Paula R. Strawser **

October 1984

ABSTRACT

Office automation systems are growing, both in use and in complexity. The development of a database management system for the office automation environment becomes a high priority, in order to provide an efficient and reliable way to manage the information needs of the office. Therefore, the specification of an 'ideal' database server for the office automation environment becomes a key area of concern. In addition to providing traditional database support, the ideal database server must also provide new database support, in order to meet the unique and many needs of office automation environments. In this paper, we focus on the characterization and specification of an ideal database server for the office automation environment. We also consider how such an ideal database server can be effectively integrated into the office automation environment. Further, we examine an experimental database system, known as the multi-backend database system (MBDS), as a candidate for the ideal database server in the office automation environment.

---

* The work reported herein is supported by Contract N00014-84-WR-24058 from the Office of Naval Research and conducted at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, CA 93943.

** S. A. Demurjian and D. K. Hsiao are with the Laboratory for Database Systems Research, Department of Computer Science, Naval Postgraduate School, Monterey, California, 93943. D. S. Kerr is with the Department of Computer Science, The Ohio State University, Columbus, OH, 43210. P. R. Strawser is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 10598.

1. INTRODUCTION

As office automation systems (OAS) become more prevalent in the work place, the need for database support in the office automation environment (OAE) becomes a key issue. In this paper we attempt to provide the characterization of an ideal database server for OAEs. The database server is used to provide traditional as well as new database support in the OAE. In addition, we study various approaches to the integration of the database server into an OAE. In our characterization and study of an ideal server, we focus on the use of an experimental database system, known as the multi-backend database system (MBDS), as the server. Although MBDS may be far from ideal, it does serve as a benchmark for measuring the other database servers for OAEs. In the rest of this paper we examine how and why MBDS may be considered as a database server for the OAE.

More specifically, in Section 2 we discuss the architecture and characteristics of an ideal database server for the OAE. In Section 3 we briefly describe the design and implementation of MBDS. In Section 4 we analyze how a database server such as MBDS can be integrated into the OAE. The analysis focuses on the multiple backend architecture of MBDS and how it does satisfy the architectural requirements of the ideal OAE database server. In Section 5, we analyze whether the unique design characteristics of MBDS meet the needs of the OAE. Finally, in Section 6 we conclude this paper.

2. A CHARACTERIZATION OF AN IDEAL DATABASE SERVER

When characterizing an ideal database server for the OAE, we focus our efforts in two directions. First, we consider the architectural requirements of the ideal database server that will facilitate the smooth integration of the ideal database server into the OAE. Second, we consider the necessary database system features or characteristics of the ideal database server for the OAE. In the following two sections, we examine these two considerations for an ideal database server in the OAE.

2.1. The Architectural Requirements

The basic structure of the OAE consists of a group of workstations connected using a local-area network (LAN) (see Figure 1, where a workstation is denoted with the letter W in a square box). To successfully meet the needs of this environment, the ideal database server must be integrated into the exist-

ing OAE. The integration of the ideal database server into the OAE must be smooth and have no ill-effect on the existing OAS. If the ideal database server runs on a single workstation, it must be powerful enough to meet the database management needs of the current and future OAE. Thus, it seems logical that the ideal database server should consist of initially a few workstations and later a number of workstations. With multiple workstations, the ideal database server should reduce and distribute the database management load across the multiple workstations.

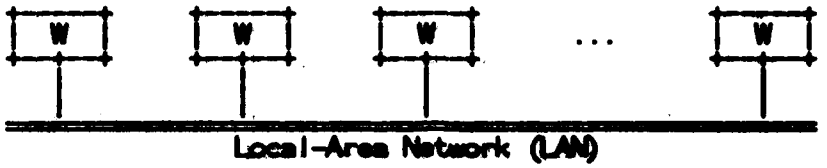


Figure 1. The Basic OAS

Whether the workstations, which make up the ideal database server act as individual database systems, or cooperate to handle the database management needs of the OAE, is also an issue. It may not be feasible in a given OAE to distribute the database management functionality and load among different database servers on the same network, since the OAS is not a distributed database system. The OAE may require a central repository of data and programs, that is maintained and accessed via a single system, so that the data and programs can be successfully shared throughout the OAE. Overall, the needs of the OAE become a crucial concern when specifying the architectural requirements of the ideal database server. In this consideration, an ideal database server for OAEs should be configured as a centralized database system running on multiple workstations.

2.2. The Six Characteristics

There are six major characteristics of an ideal database server. They are software portability, software independence, auto-configurability, survivability, versatility, and performance. Software portability provides the ideal database server with the ability to be accessible on a wide range of hardware systems. Specifically, the ideal database server should not be restricted to a particular class of hardware and a specific type of operating system. Instead, it should be portable across a wide range of workstations and operating systems of the OAE. If the ideal database driver is implemented on

multiple workstations, the software components of the driver running on the separate workstations should be sufficiently independent, so that the ideal database server will not become inoperative when a node (i.e., either one of the software components on a workstation or a workstation) becomes disabled. Software independence among system components running on separate workstations may eliminate software and hardware interdependencies and the complexity of the ideal database server.

When running on multiple workstations, the ideal database server should be auto-configurable and reconfigurable. When the OAE grows, i.e., the number of workstations in the OAS increases, or a workstation becomes disabled, the ideal database server should be able to adjust itself for the addition or loss of workstations. Such adjustment should require no new programming and no modification to the existing software drivers. Further, it should incur no disruption of the OAE or OAS. The ideal database server should also maintain a consistent and up-to-date copy of the database. When a node in the OAE is disabled, it is imperative that the ideal database server still be functional, providing continuous, albeit limited, access to the remaining database. This is also the survivability of the ideal database server.

The ideal database server should also be versatile, providing the user with more than one way of accessing the database. In an OAE where there is a large group of individuals from diverse backgrounds and with different experiences in using database facilities, the ideal database server should provide different database language interfaces in order to facilitate the database user with various ways of accessing the database. Finally, the ideal database server should be a database system that is oriented towards providing a substantial level of performance. As time goes by, both the use of the ideal database server will increase and the data and programs being stored in the database will increase. To meet the growing needs of the OAE the ideal database server must be able to expand as the OAE expands, and either maintain or increase its performance.

3. THE NEED OF A DATABASE DRIVER WITH MULTIPLE BACKEND CONFIGURATIONS

3.1. The Proposed Architecture for an Ideal Database Driver

We advocate that the architecture of an ideal database driver is configured with one controller and multiple backends. As shown in Figure 2, the controller and the backends are connected by a broadcast bus. When a

transaction is received from the host computer, the controller broadcasts the transaction to all the backends. Each backend has a number of dedicated disk drives. Since the database is distributed across the backends, a transaction can be executed by all backends concurrently. Each backend maintains a queue of transactions and schedules queries for execution independent of the other backends, in order to maximize its access operations and to minimize its idle time. On the other hand, the controller does very little work. It is responsible for receiving and broadcasting transactions, routing results, and assisting the backends in the insertion of new data. The backends do all the database operations. Just how this architecture may have the six characteristics of an ideal database server will be expounded in the following sections by way of an experimental database system which also has a similar architectural configuration.

3.2. The Multi-Backend Database System (MBDS) as a Database Driver

To provide a centralized database system, MBDS uses one or more identical minicomputers and their disk systems as database backends and a minicomputer as the database controller to interface with multiple, dissimilar workstations or mainframes. We shall refer to these workstations and mainframes as hosts or host computers. User access to the centralized database is therefore accomplished through a host computer which in turn communicates with the controller. Multiple backends are configured in parallel. The original design and analysis of MBDS are due to J. Menon [Hsia81a, Hsia81b]. An overview of MBDS can be found in [He83], with an analysis of the message-passing structure in [Boyn83a]. The implementation and new design efforts are documented in [Boyn83b, Demu84, Kerr82]. The database is distributed across all of the backends. The database management functions are replicated in each backend, i.e., all backends have identical software and hardware. They of course have different portions of the database.

There are some key issues to explore when considering MBDS for OAEs. The current implementation of MBDS uses minicomputers for both the controller and the backends. The original intent of the design was to implement a system which utilizes microprocessor-based computer systems, winchester-type disks and an Ethernet-like broadcast bus. Unfortunately, these were not available when the implementation of MBDS began in 1980. There are a number of reasons for preferring microprocessor-based computer systems or workstations over the traditional minicomputers. First, the 32-bit microprocesser is quickly

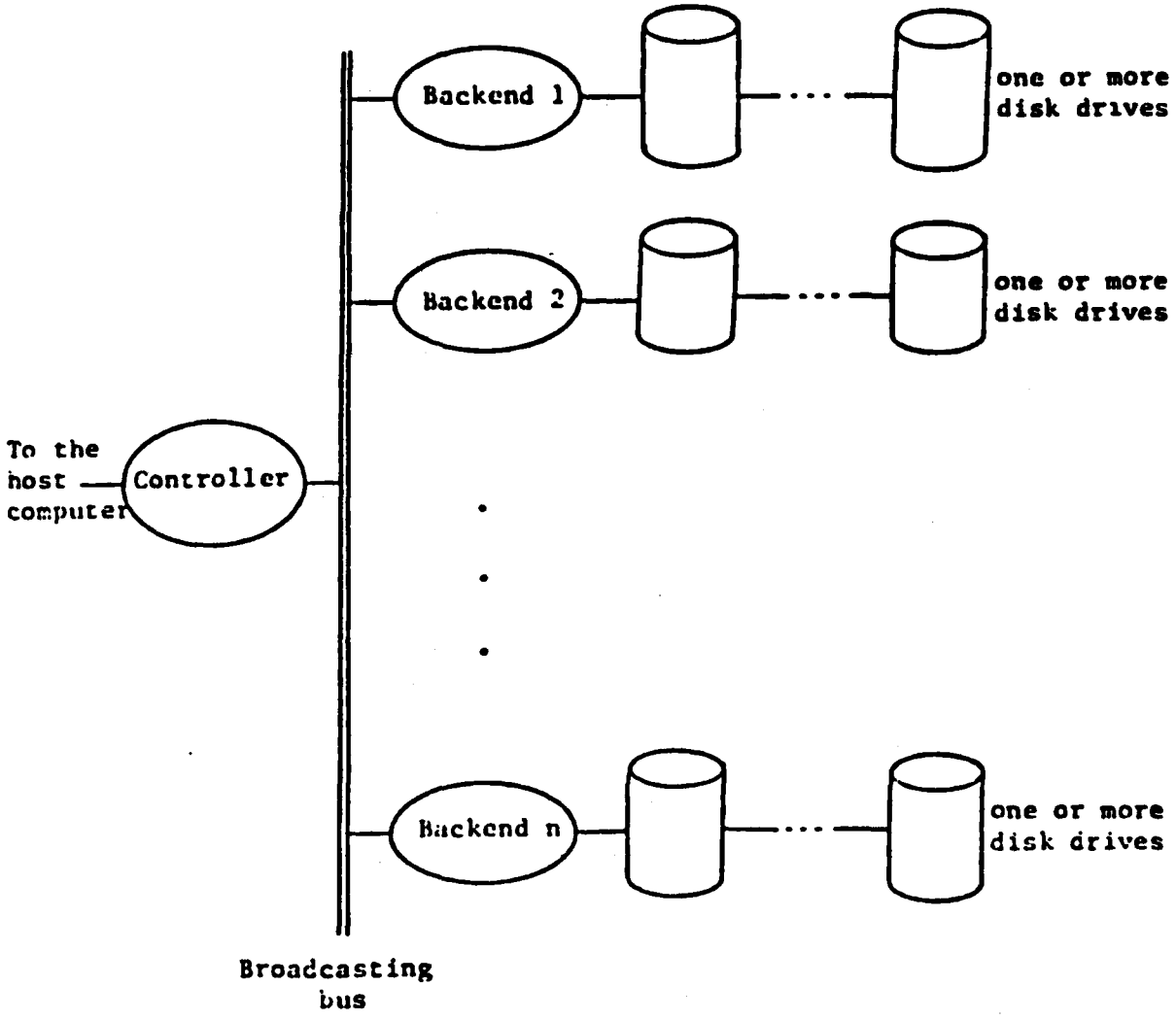


Figure 2. The MDBS Hardware Organization

attaining a reputation as a dependable, versatile and fast computer system, approaching the speed and performance of the minicomputers of five years ago.

Second, the microprocessor-based system is a cost-effective computer system. This is important when considering that MBDS requires a minimum of two computer systems. It also implies that MBDS can be expanded with relative ease and minimal cost by the addition of backend microprocessor-based computer systems.

The placement of the user interface is also affected by the use of microprocessor-based computer systems. The user interface provides access to MBDS and is run from either a separate host computer system, or as part of the system on the backend controller. When the user interface is on a separate host computer, the interface interacts with the controller via a bus. In either case, the use of a similar (with respect to the controller and backend hardware) microprocessor-based computer system for the user interface increases the compatibility and the maintainability (with respect to the hardware maintenance complexities and costs) of the database system.

The final major issue involves the ability of MBDS to support multiple data model/language interfaces to the multi-backend database system (see Appendix A). These multiple model/language interfaces allow the user to access MBDS using the relational model/SQL language, the hierarchical model/DL/1 language, the entity relationship model/Daplex language, or the network model/CODASYL language. These interfaces are also running on either a separate host computer or the backend controller; and, as such, the issues concerning the user interface also apply here.

One final note, in Appendix A, we provide a more detailed discussion of the attribute-based data model, the attribute-based data language (ABDL), the MBDS process structure, the system configurations (present and future), and the multi-lingual capabilities of MBDS.

4. FIVE APPROACHES TO THE INTEGRATION OF MBDS INTO THE OAE

In this section, we examine how MBDS can be integrated into the office automation environment. Our main focus is on ways to integrate MBDS into the OAE, and the relative advantages/disadvantages of the integration configurations. Recall that the basic OAS, consists of a group of workstations, connected by a local-area network (LAN) such as an Ethernet [Metc76]. Such a

design was shown in Figure 1. We now consider the integration of MBDS into the OAS. We approach the integration in five distinct ways.

In the first approach, MBDS is added on as a separate group of workstations in the OAS, with its own LAN. We characterize this approach as the non-integrated dual-LAN design. In this approach, the additional workstations are dedicated to the database management operations. As such, they are inaccessible for non-database activities. We provide the interface process, (which may include one or more language interfaces) as part of the user-accessible workstation. The resulting OAS is shown in Figure 3. In this and the remaining four approaches, the placement of the interface software (i.e., the number of workstations and which workstations have the interface software) is left to the discretion of the database administrator.

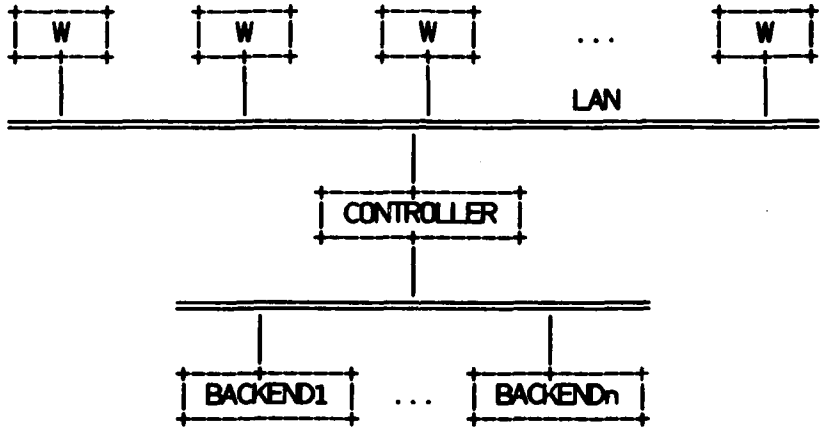


Figure 3. The Non-Integrated Dual-LAN Design

The second approach is the non-integrated single-LAN design. In this approach, as shown in Figure 4, MBDS and the OAS share a common LAN. However, the MBDS controller and backend processors still remain as separate computer systems in the OAE.

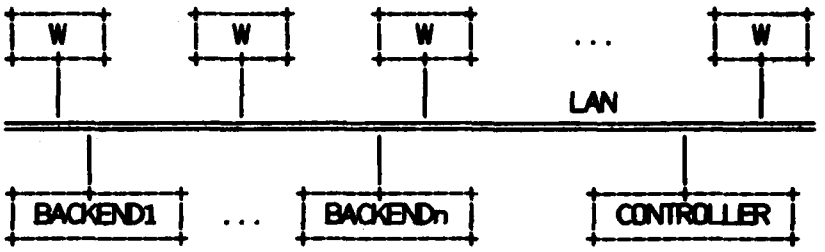


Figure 4. The Non-Integrated Single-LAN Design

The third approach, the partially-integrated design, integrates the backend processes as permanent background processes into some of the OAS workstations. The remainder of the MBDS backends are implemented as user-inaccessible workstations. The mix of the distribution of the backend processes within the user workstations is controlled by the database administrator in the OAE. The controller is the key component in MBDS, and should be devoted to overseeing the management of the database system. Therefore, the controller software is placed in a separate workstation, that is not directly utilized in the OAS. The partially-integrated design is shown in Figure 5.

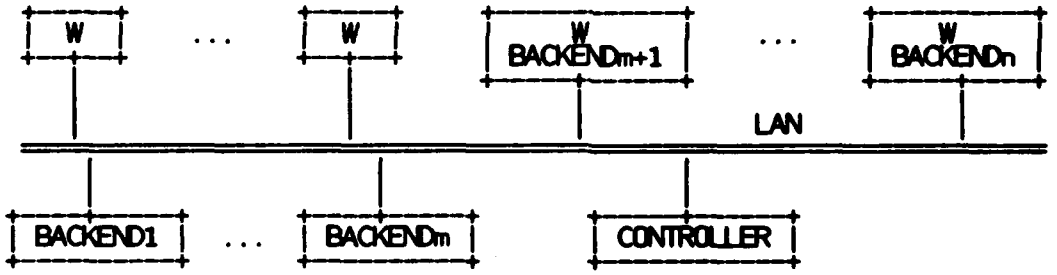


Figure 5. The Partially-Integrated Design

In the fourth approach, the isolated-controller design, the MBDS backend software is integrated into the existing workstations. As in the partially-integrated design, the controller processes are implemented in a user-inaccessible workstation. The backend processes are installed as permanent background processes in one or more workstations. The isolated-controller design is shown in Figure 6.

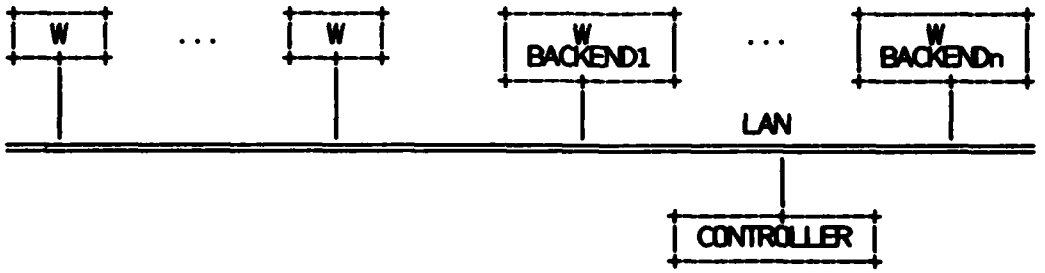


Figure 6. The Isolated-Controller Design

In the fifth approach, the fully-integrated design, the MBDS software is completely integrated into the OAS. The controller processes are installed as permanent background processes on one workstation. The backend processes are installed as permanent background processes on one or more workstations. The fully-integrated design is given in Figure 7.

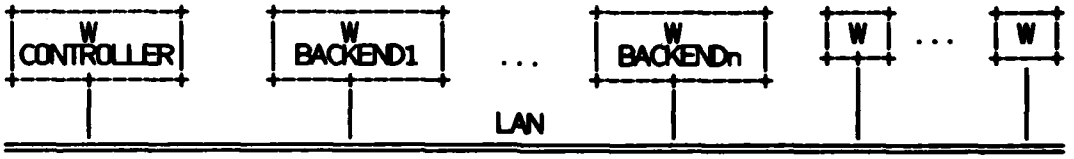


Figure 7. The Fully-Integrated Design

In the non-integrated dual-LAN design, we are using the OAS LAN as a logical two-way communications device for MBDS. Messages are passed from the interface process of a particular workstation to the controller and from the controller back to the interface process. In the remaining four designs, we are using the local area network as a logical five-way communications device. Messages are passed from the interface process to the controller, from the controller to the backends, between the backends, from the backends to the controller, and from the controller back to the interface process.

The trade-offs from one approach to the next depend on various performance and cost considerations. The non-integrated approaches differ only by the cost of an LAN, but the corresponding performance gains of the dual-LAN approach probably outweighs the cost of the extra LAN. In particular, the burden on the LAN for the OAS is significantly lower in the dual-LAN design. However, in both these approaches, a high price is paid as the database and transactions of MBDS grow in size and intensity. The integration of more backends into MBDS is costly, since the new workstations are only accessible

to the database management system.

In such a situation, either the partially-integrated design or the isolated-controller design are feasible alternatives. In both cases, keeping the controller on a non-accessible workstation is a big performance plus. In the partially-integrated design, as the database size grows, more user workstations can be configured into the database system. Further, in both cases when all backends are being used as backends for MBDS, additional workstations can be added to either system. In the partially-integrated design, those workstations can be added as either dedicated database processors or user workstations. Again, in both cases, the addition of more backends into MBDS is more cost-effective, if the backends are added as user workstations. We feel that the fully-integrated design is the least desirable. The controller as part of a user-accessible workstation would substantially degrade the performance of MBDS as the non-database use of the workstation at which the controller resides increases.

Overall, the non-integrated dual-LAN design may yield the highest performance (see Figure 3 again). The performance of the non-integrated single-LAN and partially-integrated designs are about the same. However, the partially-integrated design is more versatile and cost-effective. The isolated-controller design exhibits a moderate performance capability, but excels as a cost-effective alternative. Finally, while the fully-integrated design is cost-effective, its performance may leave a lot to be desired.

5. SIX CHARACTERISTICS OF MBDS FOR AN EFFECTIVE ROLE IN THE OAE

Regardless of the integration approach chosen, MBDS exhibits certain characteristics that are desirable in the OAE. These characteristics include the software portability of the MBDS code, the software independence of the backend code, the auto-configurability and reconfigurability of MBDS on account of its use of identical workstations and replicated software, the survivability of the system resulting from the use of duplicated directory data, the versatility of system due to the ability of MBDS to support multiple language interfaces, and the performance capabilities of the system as a result of its parallel configuration and round-robin data placement. Each of these topics is examined in the following sections.

5.1. Software Portability

The MBDS processes, i.e., the controller processes, the backend processes, and the interface process, are all written using the C programming language. C was chosen as the programming language for MBDS because of its portability, and its reputation as a good systems programming language. We estimate that the code of MBDS is about ninety-five percent portable, consisting of 13,000 lines of C code. The five percent of system-dependent code involves the inter-process message-passing code on both the VAX and the PDP-11/44s, the inter-computer message passing code for the GET and PUT processes, and the disk I/O routines for the record processing process. Thus, the great majority of the code is portable. In fact, some of the implementation development for MBDS takes place on the a VAX-11/780 running the Unix operating system, where we are able to take advantage of the C-tools provided by Unix. Thus, we feel that we have designed a relatively portable database system, that can be implemented on a wide range of the 32-bit micro-computers on the market today, e.g., the DEC MicroVAX, the Sun Workstation, etc.

5.2. Software Independence

In examining the software independence issue, we focus on the backend processes. The elegance of MBDS is that the backend software of one backend is identical to the backend software of another backend. For logical reasons, the directory data, used by each backend when processing requests, is nevertheless duplicated at every backend. However, the directory data is usually a small percentage of the non-directory data. Furthermore, the only sharing of information by the backends occurs in one phase of the directory search. Otherwise, the directory management, the concurrency control, and the record processing processes are independent of each other. So, when a new backend is configured into the system, the software present on one backend is simply replicated on the new backend. Additionally, the directory data, duplicated at an existing backend, is loaded into the new backend. When bringing a new backend into MBDS, we must also decide on whether to rearrange the non-directory data. On the one hand, we can redistribute all of the non-directory data across the disk systems of every backend. This involves reloading the data. On the other hand, we can simply leave the data undisturbed, loading only new data on the new backend. The choice is left to the discretion of the database administrator.

5.3. Auto-Configurability

One of the most convenient features of MBDS is the ability to automatically configure and reconfigure the system with ease. When starting the system for the first time, the database administrator simply specifies, using the interface, the number of backends in the system. MBDS then configures itself by notifying the controller and backend processes the number of backends on the system. Using this unique feature, MBDS can be reconfigured when a backend becomes inoperable. In such a situation, MBDS is configured with one less backend. Conversely, when a new backend is added to the system, the system can be configured with one more backend easily.

5.4. Survivability

MBDS contains only one copy of the non-directory database. When the database is loaded, it is distributed evenly across all backends' disk systems. However, the directory data, which contains index and cluster information on all data in the database, is duplicated in every backend. The distributed directory data, coupled with the software independence and reconfigurability of MBDS, offers an increased survivability of the database system in the OAE. If a backend or backends become inoperable, the system is still usable. While a backend is inoperable, a log of transactions that modified both the directory and the non-directory data is kept. When the backend is reconfigured into MBDS, the log is run for the purpose of updating the directory and other data. Although portions of the non-directory data become inaccessible with the inoperable backends, MBDS can still access and retrieve the rest of the data. Incomplete data is better than no data, provided that the user is informed of the situation.

5.5. Versatility

One of the biggest advantages of having MBDS as part of an OAS is the ability of MBDS to provide support for multiple data models (and therefore data languages) through the use multiple language-based interfaces. In the OAE, where users are from a varied range of backgrounds, such a utility is a unique feature in a database management system. In fact, the language interfaces can be tailored by the workstation. One workstation could have a SQL interface, another a DL/I interface, a third a Daplex interface, and perhaps still another have a CODASYL interface. By tailoring the language interfaces by workstation, the software required for each interface process could be

reduced. Conversely, with a wide range of language interfaces available at every workstation, the workstation becomes more accessible to a wide range of users.

5.6. Performance

The performance capabilities of any DBMS are important in an OAE, since the DBMS tends to serve as a repository of all the permanent data and programs of the OAE. As the repository becomes large and the database activities increase, the DBMS as a database server may become the performance bottleneck. However, MBDS is specifically designed to provide for capacity growth and performance enhancement. The performance metric of major concern is the response time of a request. The response time of a request is the time between the initial issuance of the request and the receipt of the final results for the request. MBDS has two original design goals. First, if the database capacity is fixed and the number of backends is increased, then the response time per request reduces proportionately. For example, if a request had a response time of 60 seconds when there is one backend, the same request would have a response time of nearly 30 seconds when there are two backends, and of nearly 15 seconds when there are four backends, provided that the database size has remained constant.

The second goal is stated that for the same requests, if the response sets are increased due to an increase of the database size and the number of backends is increased in proportion to the increase of response set, then the response time per request remains the same. For example, if a request had a response time of 60 seconds when there is one backend with 1000 records in the response set, then the same request would have a response time of close to 60 seconds when there are two backends and 2000 records in the response set. The underlying concept in each goal is that MBDS in the OAE would supply a database system that would grow as the OAS grows, and would either increase or maintain a constant response time per request by 'growing' its backends or half a given response time per request by 'doubling' its backends. On the basis of our preliminary analysis, the operational MBDS can indeed meet the two goals. The analysis is also documented in [Teka84].

6. CONCLUSIONS

We have shown how MBDS can play an important role in the OAE as an Specifically, we have shown how MBDS can provide both traditional and new

database support. We have also shown why and how MBDS should be integrated into an OAS, i.e., what MBDS has to offer to an OAE. In particular, MBDS can be integrated into an OAS in a number of ways, depending upon the needs of the office automation environment. Once MBDS is configured in an OAS, the reconfigurability feature, coupled with the replicated backend software structure, permits the database system to grow as the needs of the office information grow. Additionally, when MBDS expands, the response time per request for the system decreases proportionately, as long as the database size remains constant. As the database grows in size so grow the responses, MBDS can maintain the response time for the same type of database services by expanding its backends.

As a multi-lingual database system, MBDS offers the ability to access the database using a variety of language interfaces. From the basic data language, ABDL, to sophisticated data languages such as SQL, Daplex, CODASYL and DL/I, the user can select a language to query the database. The high degree of software portability exhibited by MBDS, allows the system to be implemented on a wide range of micro-computers, offering database support for a wide range of office automation systems. Overall, we feel that MBDS is an ideal database system for the office automation environment.

REFERENCES

[Bane78a] Banerjee, J. and Hsiao, D. K., "A Methodology for Supporting Existing CODASYL Databases with New Database Machines," Proceedings of National AOM Conference, 1978.

[Bane78b] Banerjee, J., Buam, R. I. and Hsiao, D. K., "Concepts and Capabilities of a Database Computer," AOM Transactions on Database Systems, Vol. 4, No. 1, December 1978.

[Bane80] Banerjee, J., Hsiao, D. K., and Ng, F., "Database Transformation, Query Translation and Performance Analysis of a Database Computer in Supporting Hierarchical Database Management," IEEE Transactions on Software Engineering, March 1980.

[Boyn83a] Boyne, R., et al., "A Message-Oriented Implementation of a Multi-Backend Database System (MDBS)," in Database Machines, Leillick and Missikoff (eds), Springer-Verlag, 1983.

[Boyn83b] Boyne, R., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management," Technical Report, NPS-52-83-003, Naval Postgraduate School, Monterey, California, March 1983.

[DEC79] "PCL11-B Parallel Communication Link Differential TDM Bus," Digital Equipment Corp., Maynard, Mass., 1979.

[Demu84] Demurjian, S. A., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part IV - The Revised Concurrency Control and Directory Management Processes and the Revised Definitions of Inter-Process and Inter-Computer Messages" Technical Report, NPS-52-84-005, Naval Postgraduate School, Monterey, California, March 1984.

[He83] He, X., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part II - The Design of a Prototype MDBS," in Advanced Database Machine Architecture, Hsiao (ed), Prentice Hall, 1983.

[Hsia70] Hsiao, D. K., and Harary, F., "A Formal System for Information Retrieval from Files," Communications of the AOM, Vol. 13, No. 2, February 1970, Corrigenda, Vol 13., No. 3, March 1970.

[Hsia81a] Hsiao, D. K. and Menon, M. J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC-TR-81-7, The Ohio State University, Columbus, Ohio, July 1981.

[Hsia81b] Hsiao, D. K. and Menon, M. J., "Design and Analysis of a Multi-Backend Database System for performance Improvement, Functionality Expansion and Capacity Growth (Part II)," Technical Report, OSU-CISRC-TR-81-8, The Ohio State University, Columbus, Ohio, August 1981.

[Kerr82] Kerr, D. S., et al., "The Implementation of a Multi-Backend Database System (MDBS): Part I - Software Engineering Strategies and Efforts Towards a Prototype MDBS," Technical Report, OSU-CISRC-TR-82-1, The Ohio State University, Columbus, Ohio, January 1982.

[Macy84] Macy, G., "Design and Analysis of an SQL Interface for a Multi-Backend Database System" Master's Thesis, Naval Postgraduate School, Monterey, California, March 1984.

[Metc76] Metcalfe, R. M., and Boggs, D. R., "Ethernet:Distributed Packet Switching for Local Computer Networks," Communications of the AOM, Vol. 19, July 1976.

[Muld84] Muldur, S., "The Design and Analysis of Join and Ordering Operations for a Multi-Backend Database System," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

[Teka84] Tekampe, R. C., and Watson, R. J., "Internal and External Performance Measurement Methodologies for Database Systems," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

[Weis84] Weishar, D., "Design and Analysis of a Complete Hierarchical Interface for a Multi-Backend Database System," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1984.

[Wong71] Wong, E., and Chiang, T. C., "Canonical Structure in Attribute Based File Organization," Communications of the ACM, September 1971.

APPENDIX A: THE MULTI-BACKEND DATABASE SYSTEM

In this appendix we examine the structure of the multi-backend database system, focusing on the data model, i.e., the attribute-based data model, the data language, i.e., the attribute-based data language (ABDL), the process structure, the system configurations, and the ability of MBDS to support multiple data models and database languages.

A.1 The Attribute-Based Data

In the attribute-based data model, data is modeled with the constructs: *database*, *file*, *record*, *attribute-value pair*, *directory keyword*, *directory*, record body, keyword predicate, and query. Informally, a database consists of a collection of files. Each file contains a group of records which are characterized by a unique set of directory keywords. A record is composed of two parts. The first part is a collection of attribute-value pairs or keywords. An attribute-value pair is a member of the Cartesian product of the attribute name and the value domain of the attribute. As an example, <POPULATION, 25000> is an attribute-value pair having 25000 as the value for the population attribute. A record contains at most one attribute-value pair for each attribute defined in the database. Certain attribute-value pairs of a record (or a file) are called the directory keywords of the record (file), because either the attribute-value pairs or their attribute-value ranges are kept in a directory for addressing the record (file). Those attribute-value pairs which are not kept in the directory for addressing the record (file) are called non-directory keywords. The rest of the record is textual information, which is referred to as the record body. An example of a record is shown below.

(<FILE, Census>, <CITY, Monterey>, <POPULATION, 25000>,
{ Temperate climate })

The angle brackets, <,>, enclose an attribute-value pair, i.e., keyword. The curly brackets, {,}, include the record body. The first attribute-value pair of all records of a file is the same. In particular, the attribute is FILE and the value is the file name. A record is enclosed in the parenthesis. For example, the above sample record is from the Census file.

The database is accessed by indexing on directory keywords using keyword predicates. A keyword predicate is a tuple consisting of an attribute, a relational operator (=, !=, >, <, >=, <=), and an attribute value, e.g., POPULATION >= 20000 is a keyword predicate. More specifically, it is a greater-than-or-equal-to predicate. Combining keyword predicates in disjunctive normal form characterizes a query of the database. The query

(FILE = Census and CITY = Monterey) or
(FILE = Census and CITY = San Jose)

will be satisfied by all records of the Census file with the CITY of either Monterey or San Jose. For clarity, we also employ parentheses for bracketing predicates in a query.

A.2 The Attribute-Based Data Language (ABDL)

The ABDL supports the four primary database operations, INSERT, DELETE, UPDATE, and RETRIEVE. A request in the ABDL is a primary operation with a qualification. A qualification is used to specify the information of the database that is to be operated on. Two or more requests grouped together characterize a transaction. Now, let us briefly examine the four types of requests.

The INSERT request is used to insert a new record into the database. The qualification of an INSERT request is a list of keywords which describe the record being inserted. Example 2.1 contains an INSERT request that

Example 2.1: INSERT (<FILE, Computer Science Department>,
<NAME, Hsiao>, <SALARY, 50,000>)

that will insert a record into the Computer Science Department file for the employee Hsiao with a salary of $50,000.

A DELETE request is used to remove record(s) from the database. The qualification of a DELETE request is a query. Example 2.2 is a request that

Example 2.2: DELETE ((FILE = Computer Science Department) &
(SALARY > 100,000))

would delete all records whose salary is greater than $100,000 in the Computer Science Department file.

An UPDATE request is used to modify records of the database. The qualification of an UPDATE request consists of two parts, the query and the modifier. The query specifies which records of the database are to be modified. The modifier specifies how the records being modified are to be updated. Example 2.3 is an UPDATE request that

Example 2.3: UPDATE ((FILE = Computer Science Department)
(SALARY = SALARY + 5,000))

will modify all records of the Computer Science Department file by increasing all salaries by $5,000. In this example, ((FILE = Computer Science Department)) is the query and (SALARY = SALARY + $5,000) is the modifier.

Lastly, the RETRIEVE request is used to retrieve records of the database. The qualification of a retrieve request consists of a query, a target-list, and a BY_clause. The query specifies which records are to be retrieved. The target-list is a list of output attributes. An aggregate operation, i.e., AVG, COUNT, SUM, MIN, MAX, may be applied to one or more attributes in the target-list. The optional BY_clause may be used to group records when an aggregate operation is specified. The RETRIEVE request in Example 2.4 will retrieve

Example 2.4: RETRIEVE ((FILE = Computer Science Department) &
(CITY = Monterey)) (NAME)

the employee names of all records in the Computer Science Department file with city being Monterey. ((FILE = Computer Science Department) & (CITY = Monterey)) is the query and (NAME) is the Target-List.

Obviously, ABDL is considerably more complete than the aforementioned examples have shown. For our purpose, these examples will suffice.

A.3 The Process Structure

Currently, MulBac/DBS does not communicate with a host machine. The absence of this communication requires that the test interface process, the process used to interact with MulBac/DBS, be placed in the MulBac/DBS controller. In this section we describe the process structure of MulBac/DBS. First we present the test interface process, which is used to access the system. Next, we review the processes of the controller. Finally, we describe the processes of each backend.

A.3.1 The Test Interface Process

The test interface process is a menu-driven interface to the MulBac/DBS. The main actions of the test interface are, loading a database, generating a database, and executing the request interface. When executing the request interface, the user has the option to choose a new database to work with, create a new list of traffic units, modify an existing list of traffic units, select traffic units from an existing list for execution, select an existing list so that all traffic units on the list may be executed, or specify the display mode of the results.

A.3.2 The Processes of the Controller

The controller is composed of three processes: request preparation, insert information generation, and post processing. Request preparation receives, parses and formats a request (transaction) before sending the formatted request (transaction) to the directory management process in each backend. Insert information generation is used to provide additional information to the backends when an insert request is received. Since the data is distributed, the insert only occurs at one of the backends. Thus it must determine the backend at which the insert will occur, along with certain directory information. Post processing is used to collect all the results of a request (transaction) and forward the information back to the host computer.

A.3.3 The Processes of Each Backend

Each backend is also composed of three processes. They are of course different from the controller processes. They are: directory management, concurrency control, and record processing. Directory management performs the

search of the directory structure to determine the secondary storage addresses necessary to access the clustered records. Concurrency control determines when the request can be executed. Record processing performs the operation specified by the request.

A.4 The Current and Future Configurations

The current hardware configuration of MBDS consists of a VAX-11/780 running as the controller and two PDP-11/44s running as backends. Communication between computers in MBDS is achieved by using a time-division-multiplexed bus called the parallel communication link (PCL-11B) [DEC79]. There are a total of three PCLs in the configuration, two from the VAX-11/780 to the PDP-11/44s, and one between the two PDP-11/44s. When the implementation of MBDS began in 1980, the required broadcast bus was not available. Even though we required a broadcast bus for our design, the PCL was chosen. The VAX-11/780 runs the VMS operating system, with the PDP-11/44s running the RSX-11M operating system.

The VAX-11/780 serves a dual purpose in the current configuration, as both the host computer and the controller. In addition to the controller processes described in Section 2.3, we have also implemented the interface process on the VAX. Given the large virtual and primary memory capacities of the VAX, we felt that the additional overhead of running the interface process in the controller would not be substantial. The PDP-11/44s contain only the backend processes. Plans are being made to replace the PCL-11Bs with an Ethernet-like broadcast bus and the VAX-11/780 and PDP-11/44s with microprocessor-based CPU and winchester-type disk systems, and increase the number of backends and their disk systems to six.

A.5 Supporting Multiple Language-Based Interfaces

Typically, the design and implementation of a conventional database system begins with the choice of a data model, the specification of a model-based data language, and the design and implementation of a database system which controls and executes the transactions written in the data language. Thus, we have the relational model, the SQL language and the SQL/Data System. Similarly, we have the hierarchical model, the DL/I language and the IMS system. We may also have the case of the CODASYL model, language and system. The conventional approach to the design and implementation of a system is limited to a single data model, a specific data language and a homogeneous database

system. However, the attributed-based model and the attribute-based data language of the multi-backend database system (MBDS) are sufficiently powerful and high-level and can support multiple data models and several model-based languages as if the system were a heterogeneous collection of database systems.

This unconventional design and implementation approach reveals two important database concepts. First, that the attribute-based model is an exceedingly simple yet powerful data model, such that many other data models may be realized easily by using this data model. Second, the data language of MBDS, i.e., the attribute-based data language ABDL, consists of high-level and primary operations, such that most of the other model-based language constructs can be mapped into ABDL in a straightforward fashion. There could be an SQL interface so that the transactions written in SQL can be carried out by MBDS. The execution of the transactions requires the SQL constructs to be transformed into the primary operations of ABDL through the interface. Similarly, there could be a DL/I interface so that the transactions written in DL/I can also be carried out by the interface. In this way, the single database system and multiple interfaces allow the system to support multiple data models and data languages as if it were a heterogeneous collection of database systems. In practice, we can construct a number of interfaces to support relational, hierarchical, and network operations with a minimal effort. Such an approach is clearly an attractive alternative to the approach where separate, stand-alone systems must be developed for specific models.

The procedure to construct a relational, hierarchical, or network interface to MBDS is done at both the database and data language levels. At the database level, the series of papers [Bane78a, Bane78b, Bane80] demonstrated that a relational, hierarchical, or network database can be converted into an attribute-based database. At the data language level, we focus on the development of language interfaces to the attribute-based system consistent with the user's chosen language. At this level, we address three issues. The first issue is to determine how the operations of the chosen language can be implemented using the operations of MBDS. The second issue is the translation of the language of the interface to the attribute-based data language. The third issue is the placement of the language interface within MBDS.

Our current work on language interfaces to MBDS is at the design level. The two interfaces we have designed are for SQL [Macy84, Roll84] and for DL/I

[Weis84]. To facilitate the development of the SQL interface, we also have developed algorithms to implement the sort and merge algorithms in MBDS [Muld84]. Using these designs, we plan on implementing the two interfaces in the coming months. In addition, we will be publishing a number of papers on interface development and implementation. It is sufficient to say that database support in an office automation environment should be multi-lingual in database management.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52ML Computer Science Department Naval Postgraduate School Monterey, CA 93943	10
Prof. David K. Hsiao, Code 52Hq Computer Science Department Naval Postgraduate School Monterey, CA 93943	130
Chief of Naval Research Arlington, VA 22217	1

END

FILMED

12-84

DTIC